Class 8: Web Scraping

Damian Clarke

Friday March 13, 2020

Research Methods II
MRes. in Economics

Today's Plan

Python

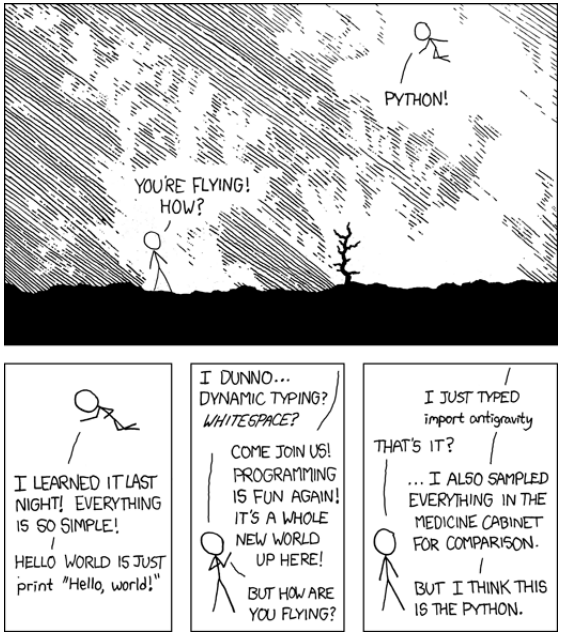
Web Scraping

Coding

Where to From Here

Why Python?

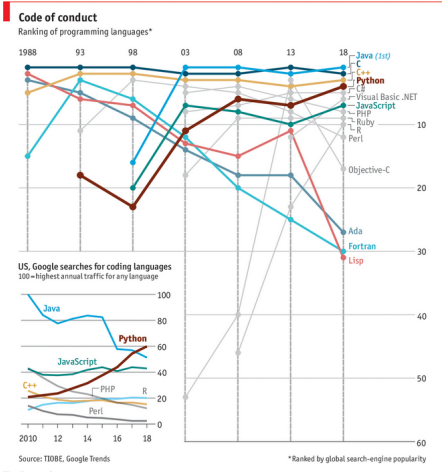
- Free
- Power over the *whole* operating system
  - Imagine if Stata had control over Firefox, image editing, Google Earth, better scientific libraries, ...
- Quite easy to get up and scraping the web (we'll do it in 20 mins)
- If you decide you like it, it can do everything for you
  - See for example Sargent and Stachurski's excellent course


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Daily chart

Python is becoming the world's most popular coding language

But its rivals are unlikely to disappear



The Economist

What Do You Need?

- Unix or OS X: nothing!
- Windows: In many distributions Python is not installed by default
  - For complete packages, install Anaconda (http://continuum.io/), or follow online tutorials for your particular OS
- Other external packages (from the Python Package Index) can be installed using `pip install package-name`
- It will be useful to install a stand alone text editor with syntax highlighting (eg Sublime, gedit, emacs, ...)
- Note that there are two versions of Python: Python2 and Python3. Today we will use Python2. Going forward, Python3 should be preferred.

How to Run Python

- A number of ways: from the command line, interactively, using ipython
- For the interests of time, we'll just run from the command line
  - However, if you're going to run this frequently, IPython and Jupytr are worth checking out
- If you're interested in following along online (without downloading Python to your local machine), go to https://repl.it/languages/Python2

What is Web Scraping?

Essentially, the process of harvesting data that is directly stored on the web in an irregular or highly disperse format.

- When undertaking econometric analysis, we of course want very regular data, formatted into lines and columns
- Generally two steps:
  - Looping through nested urls to get to (many) source html pages
  - Taking html (or some other output) and formatting into a useful structure
- This second step can download all manner of things (eg pdfs, xls, images, ...) which can then be processed computationally
- There are a number of tools people use for this sort of analysis: Python, R, RapidMiner, even MATLAB ...

Why is this useful?

- Often data is not stored directly as a csv or some other standardised format
- In some cases, data does not yet exist in any centralised form
- This opens up many entirely different types of data we mightn't have previously thought about
- The majority of economics papers are now using 'novel' data (ie not survey based)

What can we do with it?

- It has come in handy for me many times
  - Download, unzip and merge 1000+ DHS surveys, up to date at the second that scraping takes place
  - Download all (30,000+) papers on NBER for text analysis
  - Download election results: India, Philippines
  - Repeated calls to World Bank Data Bank
- And turns up frequently in economics papers, among many others:
  - "The Billion Prices Project" Cavallo & Rigobon (2016)
  - "Nowcasting the Local Economy: Using Yelp Data to Measure Economic Activity" Glaeser et al. (2017)
  - Many others (see Table 1 of Edelman (JEP, 2012))

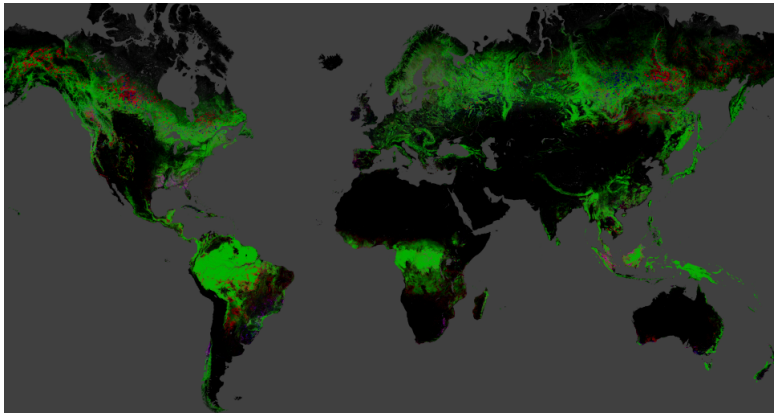


Figure: And it can look quite cool...

Hansen, M.C. et al (2013) High-Resolution Global Maps of 21st-Century Forest Cover Change. *Science* 342 (6160) 850-853.

Some Considerations

There are a number of considerations you should take into account prior to undertaking a web-scraping project:

1. Is the target standardized enough?
2. Are there legal limits?
3. Are there technical limits?

Some Considerations (Part B)

Note that some websites also have simpler ways to do this so you can avoid web-scraping. The site Ideas RePEc is a great example of this. See `https://ideas.repec.org/getdata.html`.

- Perhaps information is already provided in a central download option
- Sites may also have APIs (perhaps at a cost) to simplify tasks
- Upon request, owners of sites may be happy to provide you data

1. Is the target standardized enough?

As we will see below, webscraping requires us to interact with the source code (html or xml) of websites

- Webscraping will be considerably harder if we are trying to get data of many disperse websites rather than many pages within a single site
- It will also be more challenging of source/format of data within a site changes over time
  - Eg, in early years, pdfs which are scanned, and in later years machine-readable
- Independent of this, some websites are stored on very unstable servers

2. Are there legal limits?

Some websites *do not allow* webscraping. This may be because they consider their information proprietry, or because they wish to avoid robot traffic slowing down servers.

- The law here is complex and depends upon jurisdiction
- But there are certainly precedents suggesting it can be troubling, eg LinkedIn Corporation vs. Does, 1 through 100 inclusive
- Where possible, seek permission, read Terms of Service
- And always consider the `robots.txt` file

3. Are there technical limits?

Many websites have safeguards against excessive crawling or scraping (for example "Completely Automated Public Turing test to tell Computers and Humans Apart").

- Sometimes these will be quite simple things like limiting rate of requests or temporarily blocking IPs
- These can often be worked around by slowing the rate of webscraping
- In the more complicated case where IPs are blocked completely, you could consider looping over different IPs, eg using proxies or TOR
- This is not trivial

Coding

We will go through a relatively simple (and contrived) example.

- For this process, there are a number of tools we will use:
  - Ideally, a web browser that lets us look at source code (pretty much any of them)
  - Regular Expressions (Python's `re`)
  - If this is a big job, we should think about error capture (Python's `try` command)

An Aside on Regular Expressions

Regular Expressions (or regex) are a standardised way to search within text to match patterns.

- Most languages with string capabilities have their own regex libraries
- While precise syntax varies by language, there is a standard set of tools
- These can often be quite simple, however can become very complex (eg complete email validation regexs
- It is worth looking at syntax for language you are using, and working through examples (see here for Python syntax/examples)

Basic Code

```
1  #  Scrape_xkcd 0.01              damiancclarke              yyyy-mm-dd:2020-03-13
2  #---|----1----|----2----|----3----|----4----|----5----|----6----|----7----|----8
3  #
4
5  ********************************************************************************
6  # (1) Import required packages, set-up names used in urls
7  ********************************************************************************
8  import urllib2
9  import re
10
11 target = 'http://www.xkcd.com'
12
13 ********************************************************************************
14 # (2) Scrape target url and print source code
15 ********************************************************************************
16 response = urllib2.urlopen(target)
17 print response
```

Complete Code

```
# (1) Import required packages, set-up names used in urls
import urllib2
import re
target = 'http://www.xkcd.com'
# (2) Scrape target url and find the last comic number (num)
response = urllib2.urlopen(target)
for line in response:
    search = re.search('Permanent link to this comic:', line)
    if search!=None:
        lastcomic=re.findall('\d*', line)
for item in lastcomic:
    if len(item)>0:
        num = int(item)
# (3) Loop through all comics, finding each comic's title or capturing errors
for append in range(1, num+1):
    url = target + '/' + str(append)
    response = urllib2.urlopen(url)
    for line in response:
        search = re.search('ctitle',line)
        if search!=None:
              print line[17:-7]
```

Or, With Error Capture

```
#*****************************************************************************
# (3) Loop through all comics, finding each comic's title or capturing errors
#*****************************************************************************
for append in range(1, num+1):
    url = target + '/' + str(append)
    try:
        response = urllib2.urlopen(url)
        for line in response:
            search = re.search('ctitle',line)
            if search!=None:
                print line[17:-7]
    except urllib2.HTTPError, e:
        print('%s has http error' % url)
    except urllib2.URLError, e:
        print('%s has url error' % url)
```

Exporting Our 'Data'

Python is extremely capable at editing text to create output files:

```
#*****************************************************************************
# (3) Loop through all comics, finding each comic's title or capturing errors
#*****************************************************************************
output = open('xkcd_names.txt', 'w')
output.write('Comic, Number, Title \n')

for append in range(1, num+1):
    url = target + '/' + str(append)
    response = urllib2.urlopen(url)
    for line in response:
        search = re.search('ctitle',line)
        if search!=None:
            print line[17:-7]
            output.write('xkcd,' + str(append) + ',' + line[17:-7] + '\n')

output.close()
```

Where to From Here

- You can actually get remarkably far with Python + a web browser + Regular Expressions!
- Some times you may want a more structured approach: Beautiful Soup
  - Let's have a look at `Scrape_xkcd_bs.py` to see an example with BeautifulSoup
- Python can do much, much, much more
  - Pandas
  - NumPy
  - Matplotlib
  - SciPy
- Questions/comments?